Project Report

MATH 641: Time Series Analysis I

Fall 2024

Topic:

Bitcoin Price Prediction (Non-Seasonal Data)
Airline Passenger Traffic Prediction (Seasonal Data)

Name:

Sanjana Gupta

December 17, 2024

Contents

1 Bitcoin Price Prediction (Non-Seasonal Data)

1.1 Motivation and Introduction

Bitcoin is the most prominent cryptocurrency, characterized by high volatility and global financial influence. Predicting Bitcoin prices is crucial for traders, investors, and financial analysts to develop informed strategies and manage risks. This project aims to analyze Bitcoin's daily price trends and provide short-term forecasts to address the challenges of volatility in financial planning.

1.2 Data Preprocessing

The dataset consists of hourly Bitcoin prices aggregated into daily intervals, obtained from a reliable financial database. Key variables include:

- **Open:** The price at the start of the day.
- **High:** The highest price during the day.
- **Low:** The lowest price during the day.
- **Close:** The price at the end of the day.
- **Volume:** The total trading volume.

Data preprocessing involved aggregating hourly data, identifying and handling missing values, and converting timestamps to a readable date format. Figure 1 and 2 show the daily price trends.

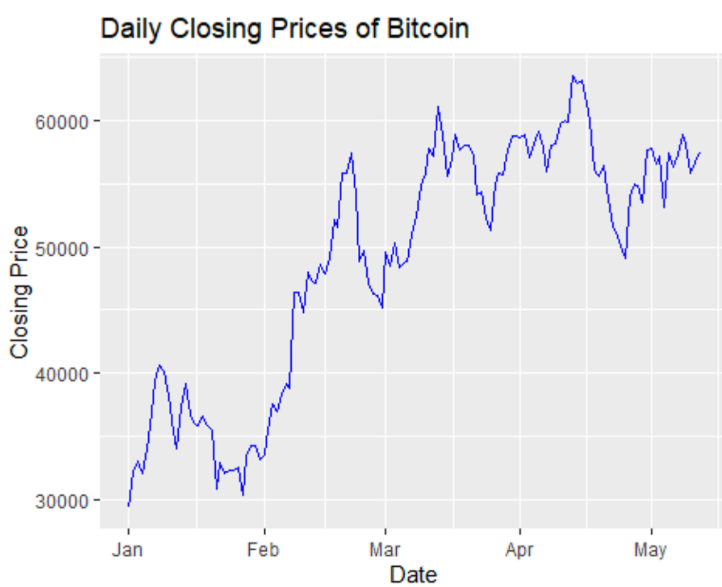


Figure 1: Bitcoin Price Trend

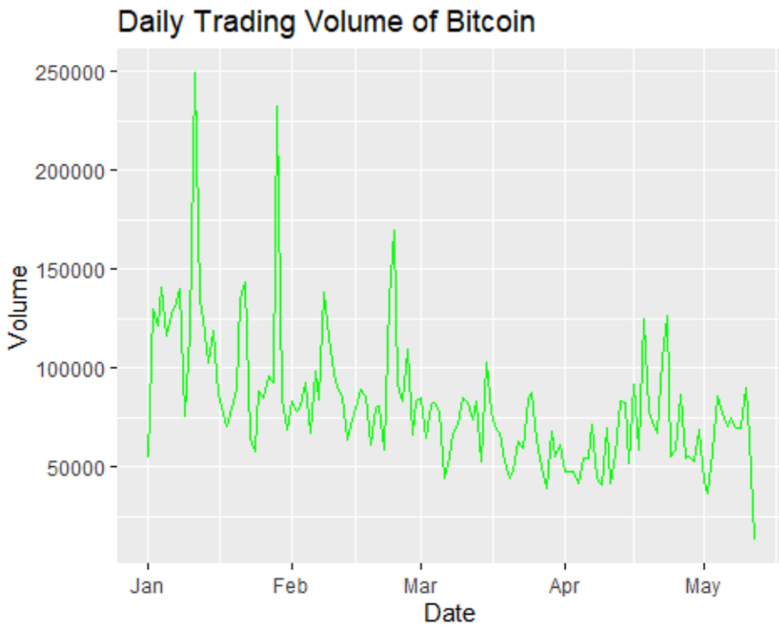


Figure 2: Daily Trading Volume of Bitcoin

1.3 Box-Jenkins Models

The Box-Jenkins methodology was applied systematically to develop an appropriate model for the time series data. The key steps involved were as follows:

Stationarity Check:

The stationarity of the raw data was tested using the Augmented Dickey-Fuller (ADF) test. The test results indicated non-stationarity in the original data. To address this, first differencing was applied, successfully transforming the data into a stationary series.

Model Identification:

The best-fitting ARIMA model was identified using the `auto.arima()` function. Based on the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC), the ARIMA(0,1,0) model was selected as the optimal model.

Model Diagnostics:

The residuals of the selected model were evaluated to ensure adequacy. Independence of residuals was confirmed through the Ljung-Box test, which showed no significant autocorrelations. This result validated the suitability of the chosen model for the data.

The Box-Jenkins approach provided a robust framework for modeling the time series and ensured that the final model met all necessary assumptions.

1.4 Forecasting

The ARIMA(0,1,0) model was used to forecast the next 30 days of Bitcoin prices. The forecast shows high volatility, consistent with Bitcoin's historical behavior. Figure 3 illustrates the forecasted prices along with confidence intervals.

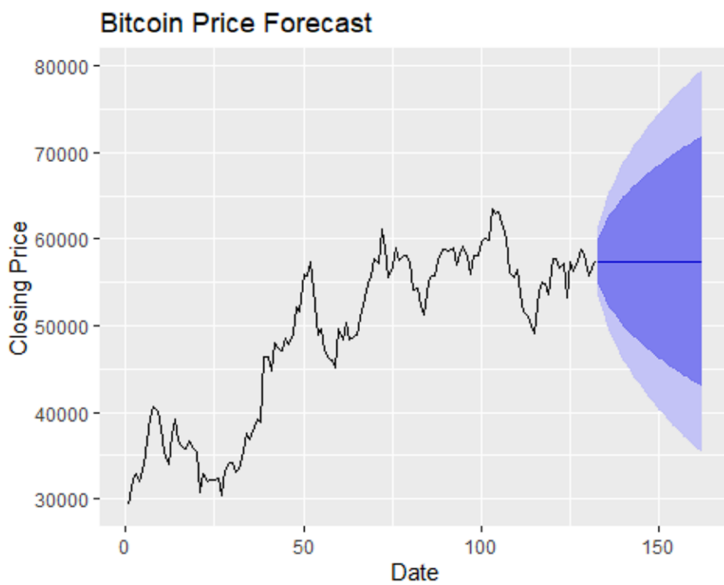


Figure 3: Bitcoin Price Forecast

1.5 Statistical Conclusions

The ARIMA(0,1,0) model effectively captured short-term trends in Bitcoin prices. Residual independence validated the model's reliability. However, the model is limited by the erratic nature of Bitcoin prices, making long-term forecasting less predictable.

1.6 Conclusions in the Context of the Problem

The forecasted Bitcoin prices provide valuable insights for short-term financial planning and risk management. Investors can use these forecasts to adjust trading strategies, but the inherent volatility highlights the importance of caution when interpreting results.

2 Airline Passenger Traffic Prediction (Seasonal Data)

2.1 Motivation and Introduction

Airline passenger traffic exhibits clear seasonal patterns, with peaks during summer and holiday seasons. Accurate forecasts are essential for airlines to optimize resources, plan schedules, and improve customer service. This project aims to analyze historical passenger data and provide seasonal forecasts to guide operational planning.

2.2 Data Preprocessing

The dataset consists of monthly international airline passenger totals from 1949 to 1960. Key attributes include:

- Month: Time of observation.
- Passengers: Number of passengers in thousands.

Figure 4 illustrates the trends in passenger traffic.

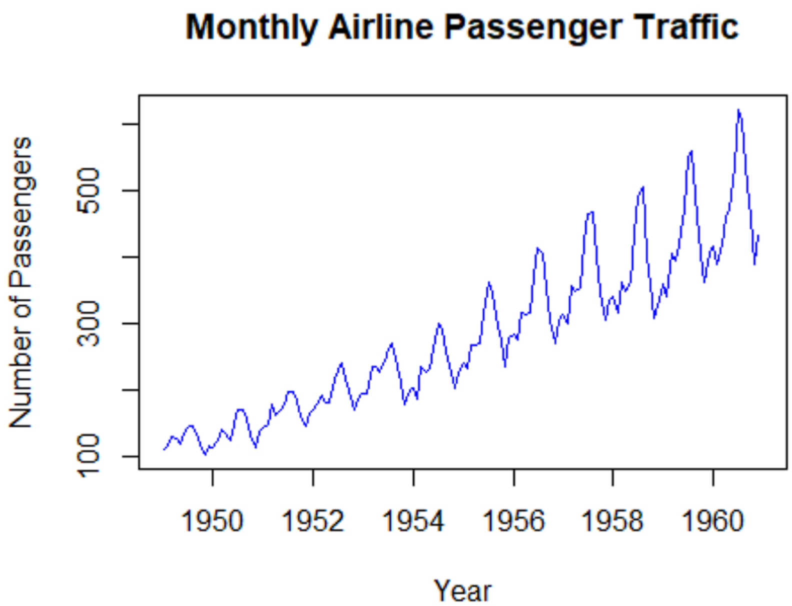


Figure 4: Monthly Airline Passenger Traffic

Figure 5 illustrates the Decomposition of Multiplicative Time Series.

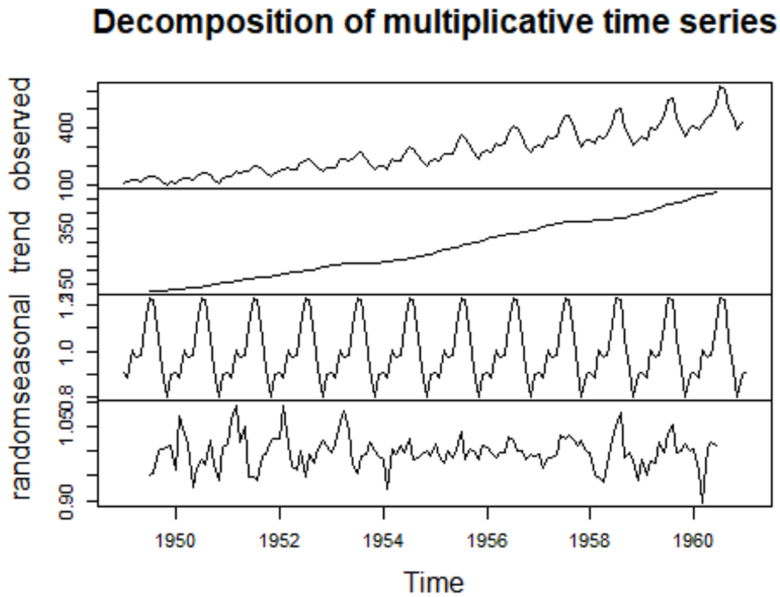


Figure 5: Decomposition of Multiplicative Time Series

2.3 Box-Jenkins Models

The Box-Jenkins methodology for seasonal data included:

Stationarity Check:

The Augmented Dickey-Fuller (ADF) tests indicated non-stationarity in the data. To address this, seasonal differencing was applied. Additionally, a log transformation was used to stabilize variance in the time series.

Figure 6 illustrates the data after log transformation.

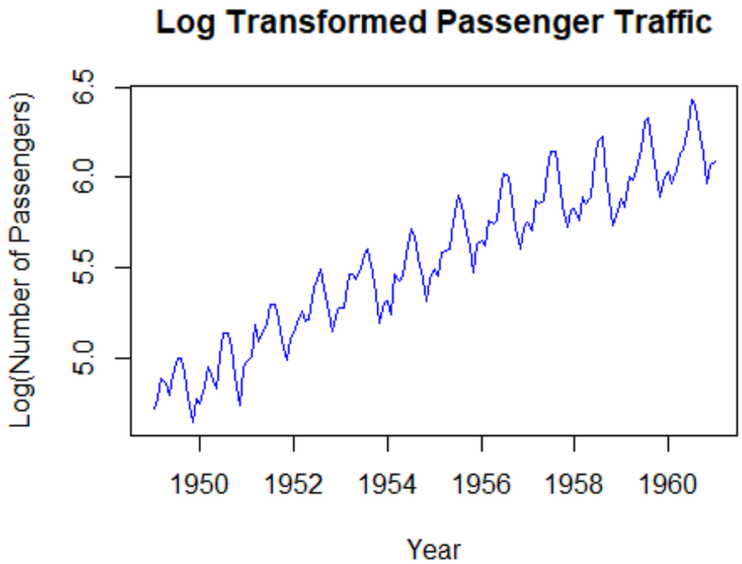


Figure 6: Log transformed Passenger Traffic

Model Identification:

The SARIMA(0,1,1)(0,1,1)[12] model was identified as the best-fitting model using the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC).

Model Diagnostics:

Residual diagnostics confirmed the adequacy of the model. The residuals passed the Ljung-Box test, indicating independence. Furthermore, residual plots revealed no significant patterns, validating the model assumptions.

Figure 7 illustrates the residual plots.

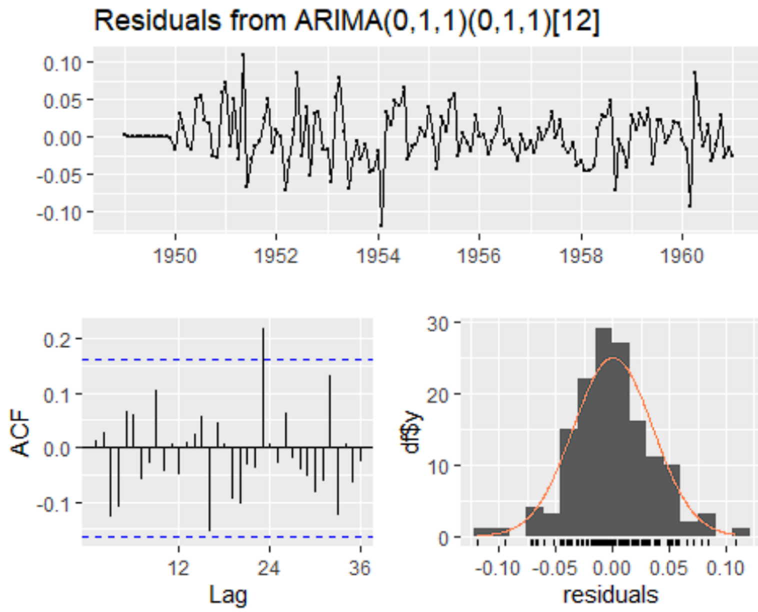


Figure 7: Residual Plots

The methodology successfully addressed stationarity, identified an optimal model, and ensured its adequacy through rigorous diagnostics.

2.4 Forecasting

The SARIMA(0,1,1)(0,1,1)[12] model forecasted the next 24 months of passenger traffic. Seasonal peaks in summer and troughs in winter were evident.

Figure 8 illustrates the SARIMA Forecast of Airline Passenger Traffic.

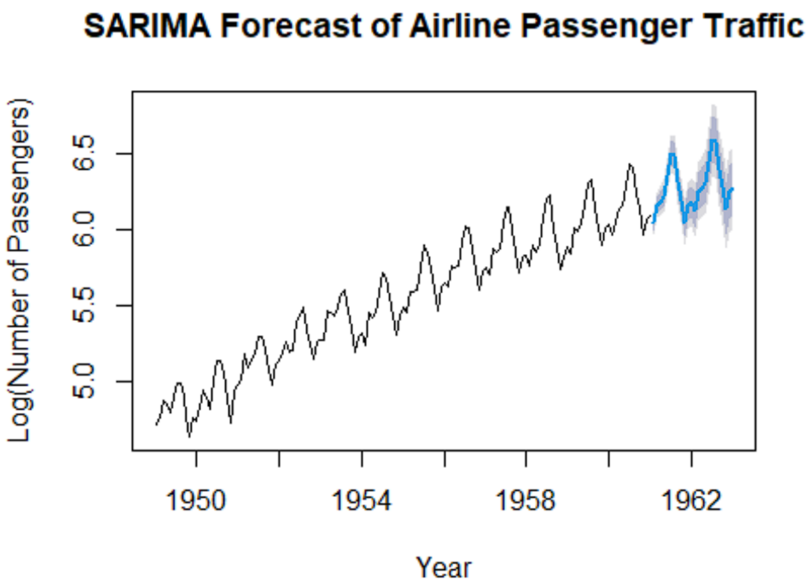


Figure 8: SARIMA Forecast

2.5 Statistical Conclusions

The SARIMA(0,1,1)(0,1,1)[12] model effectively captured the seasonal patterns in the data. Model diagnostics validated its reliability, and the forecast results aligned with expected trends, demonstrating its suitability for resource planning.

2.6 Conclusions in the Context of the Problem

The forecasts provide actionable insights for airlines, highlighting growth during peak seasons. This information can guide resource allocation, scheduling, and marketing strategies to meet customer demand efficiently.

Appendix

A Bitcoin Price Prediction - code and outputs

B Airline Passenger Traffic Prediction - code and outputs

A - Bitcoin Price Prediction

2024-12-16

```
# Load necessary libraries
library(dplyr)

## Warning: package 'dplyr' was built under R version 4.4.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(lubridate)

## Warning: package 'lubridate' was built under R version 4.4.2

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.4.2

# Read the data
data <- read.csv("C:\\Users\\USER\\Downloads\\data1\\main.csv")

# Check the structure of the data
str(data)

## 'data.frame':    188317 obs. of  11 variables:
##  $ Open.Time                    : num  1.61e+12 1.61e+12 1.61e+12 1.61e+12
1.61e+12 ...
##  $ Open                         : num  28924 28962 29010 28990 28983 ...
##  $ High                         : num  28962 29018 29017 29000 28996 ...
##  $ Low                          : num  28913 28961 28974 28972 28972 ...
##  $ Close                        : num  28962 29010 28989 28983 28976 ...
##  $ Volume                       : num  27.5 58.5 42.5 30.4 24.1 ...
##  $ Close.Time                   : num  1.61e+12 1.61e+12 1.61e+12 1.61e+12
1.61e+12 ...
##  $ Quote.asset.volume           : num  794382 1695803 1231359 880017 699226
```

```
...
##  $ Number.of.trades           : int  1292 1651 986 959 726 952 750 782
886 1558 ...
##  $ Taker.buy.base.asset.volume : num  16.78 33.73 13.25 9.46 6.81 ...
##  $ Taker.buy.quote.asset.volume: num  485391 978176 384077 274083 197519
...
```

```r
head(data)
```

```
##      Open.Time     Open     High      Low    Close   Volume   Close.Time
## 1 1.609459e+12 28923.63 28961.66 28913.12 28961.66 27.45703 1.609459e+12
## 2 1.609459e+12 28961.67 29017.50 28961.01 29009.91 58.47750 1.609459e+12
## 3 1.609459e+12 29009.54 29016.71 28973.58 28989.30 42.47033 1.609459e+12
## 4 1.609459e+12 28989.68 28999.85 28972.33 28982.69 30.36068 1.609459e+12
## 5 1.609459e+12 28982.67 28995.93 28971.80 28975.65 24.12434 1.609459e+12
## 6 1.609460e+12 28975.65 28979.53 28933.16 28937.11 22.39601 1.609460e+12
##   Quote.asset.volume Number.of.trades Taker.buy.base.asset.volume
## 1           794382.0             1292                   16.777195
## 2          1695802.9             1651                   33.733818
## 3          1231358.7              986                   13.247444
## 4           880016.8              959                    9.456028
## 5           699226.2              726                    6.814644
## 6           648322.7              952                    9.127550
##   Taker.buy.quote.asset.volume
## 1                     485390.8
## 2                     978176.5
## 3                     384076.9
## 4                     274083.1
## 5                     197519.4
## 6                     264217.9
```

Step 2: Convert Timestamps to Date-Time

```r
# Convert Open Time and Close Time to POSIXct (date-time format)
data$Open.Time <- as.POSIXct(data$Open.Time / 1000, origin = "1970-01-01", tz
= "UTC")
data$Close.Time <- as.POSIXct(data$Close.Time / 1000, origin = "1970-01-01",
tz = "UTC")

# Verify the conversion
head(data$Open.Time)
```

```
## [1] "2021-01-01 00:00:00 UTC" "2021-01-01 00:01:00 UTC"
## [3] "2021-01-01 00:02:00 UTC" "2021-01-01 00:03:00 UTC"
## [5] "2021-01-01 00:04:00 UTC" "2021-01-01 00:05:00 UTC"
```

```r
head(data$Close.Time)
```

```
## [1] "2021-01-01 00:00:59 UTC" "2021-01-01 00:01:59 UTC"
## [3] "2021-01-01 00:02:59 UTC" "2021-01-01 00:03:59 UTC"
## [5] "2021-01-01 00:04:59 UTC" "2021-01-01 00:05:59 UTC"
```

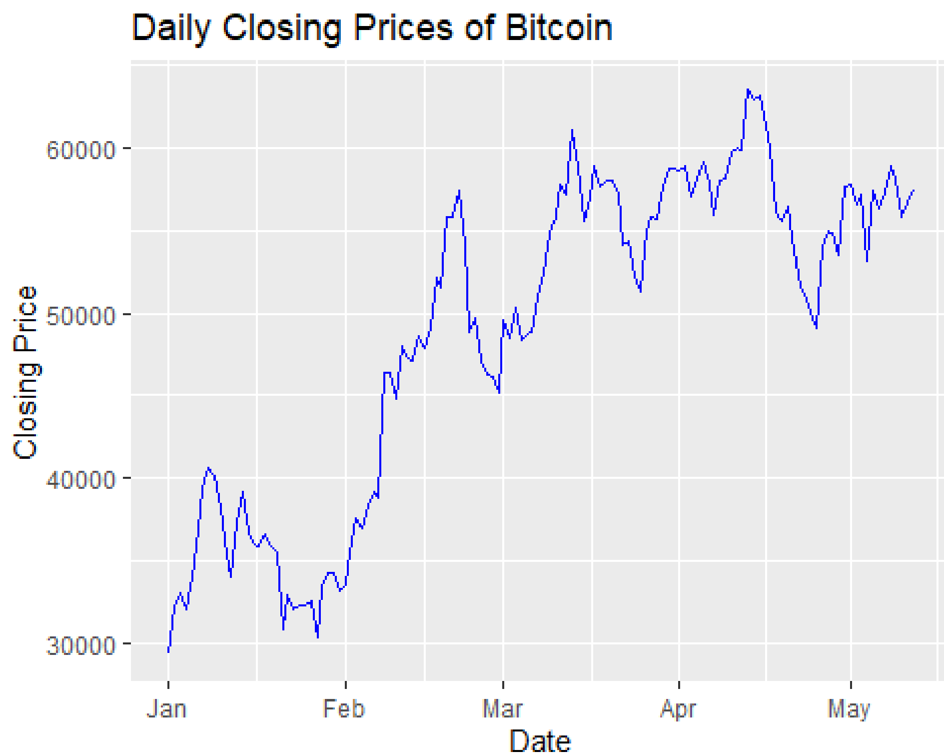
Step 3: Aggregate Data to Daily Intervals

```r
# Aggregate to daily data
daily_data <- data %>%
  mutate(Date = as.Date(Open.Time)) %>%
  group_by(Date) %>%
  summarize(
    Open = first(Open),
    High = max(High),
    Low = min(Low),
    Close = last(Close),
    Volume = sum(Volume),
    Trades = sum(Number.of.trades)
  )

# Verify the daily aggregated data
head(daily_data)
```

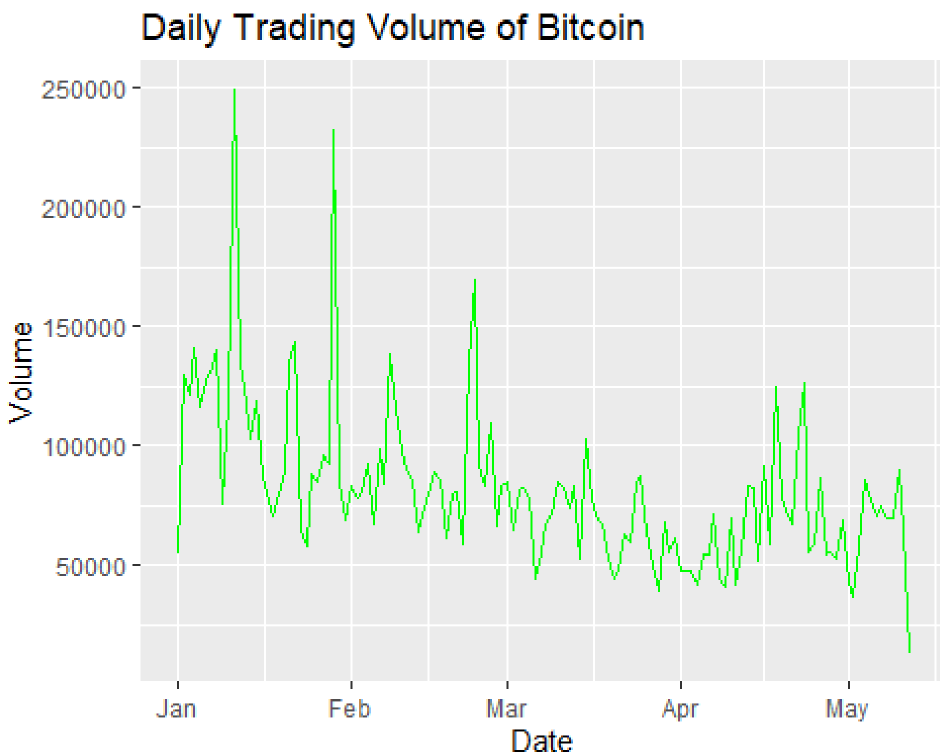
```
## # A tibble: 6 × 7
##   Date         Open   High    Low  Close  Volume  Trades
##   <date>      <dbl>  <dbl>  <dbl>  <dbl>   <dbl>   <int>
## 1 2021-01-01 28924. 29600  28625. 29332.  54183. 1314910
## 2 2021-01-02 29332. 33300  28947. 32178. 129994. 2245922
## 3 2021-01-03 32176. 34778. 31963. 33000. 120958. 2369698
## 4 2021-01-04 33000. 33600  28130  31989. 140900. 2642408
## 5 2021-01-05 31990. 34360  29900  33950. 116050. 2526851
## 6 2021-01-06 33950. 36939. 33288  36769. 127139. 2591783
```

Step 4: Visualize the Data

```r
# Plot Daily Closing Prices
ggplot(daily_data, aes(x = Date, y = Close)) +
  geom_line(color = "blue") +
  labs(title = "Daily Closing Prices of Bitcoin", x = "Date", y = "Closing
Price")
```




```r
# Plot Daily Trading Volume
ggplot(daily_data, aes(x = Date, y = Volume)) +
  geom_line(color = "green") +
  labs(title = "Daily Trading Volume of Bitcoin", x = "Date", y = "Volume")
```



Step 5: Check for Stationarity

```
# Load required library
library(tseries)

## Warning: package 'tseries' was built under R version 4.4.2

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

# Perform the Augmented Dickey-Fuller (ADF) test on Closing Prices
cat("ADF Test for Closing Prices:\n")

## ADF Test for Closing Prices:

adf_result <- adf.test(daily_data$Close, alternative = "stationary")
print(adf_result)

##
##  Augmented Dickey-Fuller Test
##
## data:  daily_data$Close
## Dickey-Fuller = -2.1389, Lag order = 5, p-value = 0.5187
## alternative hypothesis: stationary

# Check stationarity and apply differencing if needed
if (adf_result$p.value > 0.05) {
```

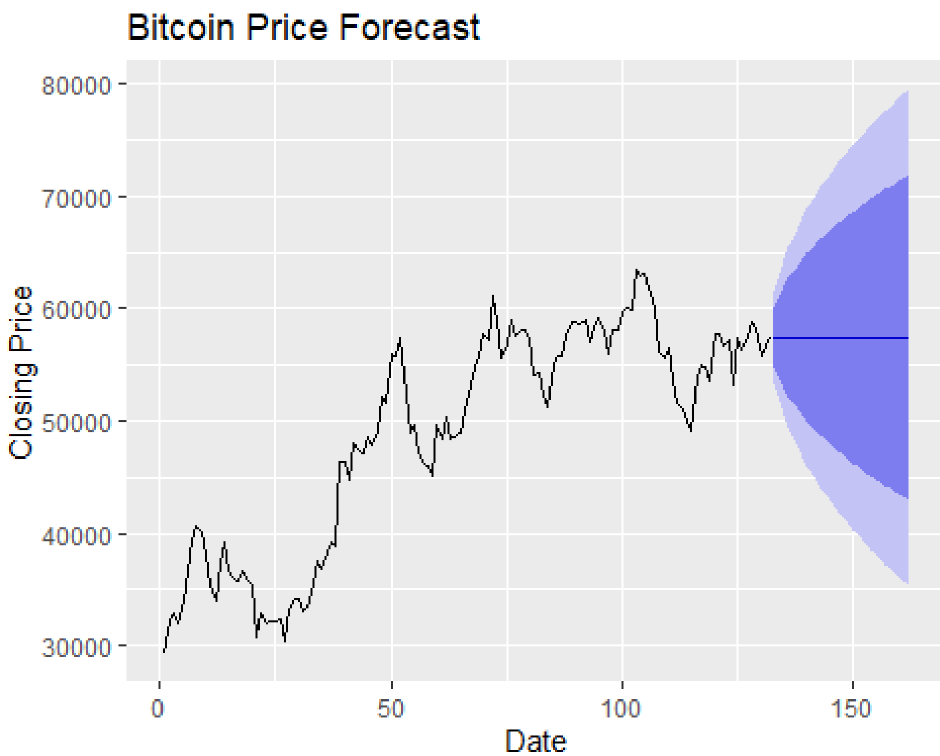
```r
  cat("\nSeries is non-stationary. Applying first differencing...\n")

  # Apply differencing and add to dataframe
  daily_data$Close_diff <- c(NA, diff(daily_data$Close))  # Prepend NA to
align rows

  # Verify the new column
  print(head(daily_data))

  # Perform ADF test on the differenced series
  cat("\nADF Test for Differenced Closing Prices:\n")
  adf_diff_result <- adf.test(na.omit(daily_data$Close_diff), alternative =
"stationary")
  print(adf_diff_result)

  # Check if stationarity is achieved
  if (adf_diff_result$p.value <= 0.05) {
    cat("\nThe differenced series is stationary.\n")
  } else {
    cat("\nThe differenced series is still non-stationary. Further
transformations may be required.\n")
  }
} else {
  cat("\nThe original series is stationary. No differencing is needed.\n")
}
```

```
##
## Series is non-stationary. Applying first differencing...
## # A tibble: 6 × 8
##   Date         Open   High    Low  Close  Volume  Trades Close_diff
##   <date>      <dbl>  <dbl>  <dbl>  <dbl>   <dbl>   <int>      <dbl>
## 1 2021-01-01 28924. 29600  28625. 29332.  54183. 1314910         NA
## 2 2021-01-02 29332. 33300  28947. 32178. 129994. 2245922      2847.
## 3 2021-01-03 32176. 34778. 31963. 33000. 120958. 2369698       822.
## 4 2021-01-04 33000. 33600  28130  31989. 140900. 2642408     -1011.
## 5 2021-01-05 31990. 34360  29900  33950. 116050. 2526851      1961.
## 6 2021-01-06 33950. 36939. 33288  36769. 127139. 2591783      2820.
##
## ADF Test for Differenced Closing Prices:

## Warning in adf.test(na.omit(daily_data$Close_diff), alternative =
## "stationary"): p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  na.omit(daily_data$Close_diff)
## Dickey-Fuller = -4.6466, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
##
```

```
##
## The differenced series is stationary.
```

Step 6: Fit ARIMA Model and Forecast

```r
library(forecast)

## Warning: package 'forecast' was built under R version 4.4.2

# Fit ARIMA Model
fit <- auto.arima(daily_data$Close, seasonal = FALSE)

# Summary of the Model
summary(fit)

## Series: daily_data$Close
## ARIMA(0,1,0)
##
## sigma^2 = 4214274:  log likelihood = -1185.02
## AIC=2372.03   AICc=2372.07   BIC=2374.91
##
## Training set error measures:
##                    ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 213.2411 2045.079 1542.794 0.4115671 3.270829 0.9925672
##                     ACF1
## Training set -0.05299381

# Forecast the next 30 days
forecasted <- forecast(fit, h = 30)

# Plot the Forecast
autoplot(forecasted) +
  labs(title = "Bitcoin Price Forecast", x = "Date", y = "Closing Price")
```


Bitcoin Price Forecast
80000
70000
60000
50000
40000
30000
Closing Price
0
50
100
150
Date

B - Airline Passenger Traffic Prediction

2024-12-16

Step 1: Load and Inspect the Dataset

```
# Load necessary libraries
library(dplyr)

## Warning: package 'dplyr' was built under R version 4.4.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.4.2

library(lubridate)

## Warning: package 'lubridate' was built under R version 4.4.2

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

# Read the data
data <- read.csv("C:\\Users\\USER\\Downloads\\SEASONAL DATA\\international-
airline-passengers.csv")

# Inspect the data
head(data)

##     Month
## 1 1949-01
## 2 1949-02
## 3 1949-03
## 4 1949-04
## 5 1949-05
## 6 1949-06
##
```

```
International.airline.passengers..monthly.totals.in.thousands..Jan.49...Dec.6
0
## 1
112
## 2
118
## 3
132
## 4
129
## 5
121
## 6
135
```

```r
# Check the structure
str(data)
```

```
## 'data.frame':    145 obs. of  2 variables:
##  $ Month
: chr  "1949-01" "1949-02" "1949-03" "1949-04" ...
##  $
International.airline.passengers..monthly.totals.in.thousands..Jan.49...Dec.6
0: int  112 118 132 129 121 135 148 148 136 119 ...
```

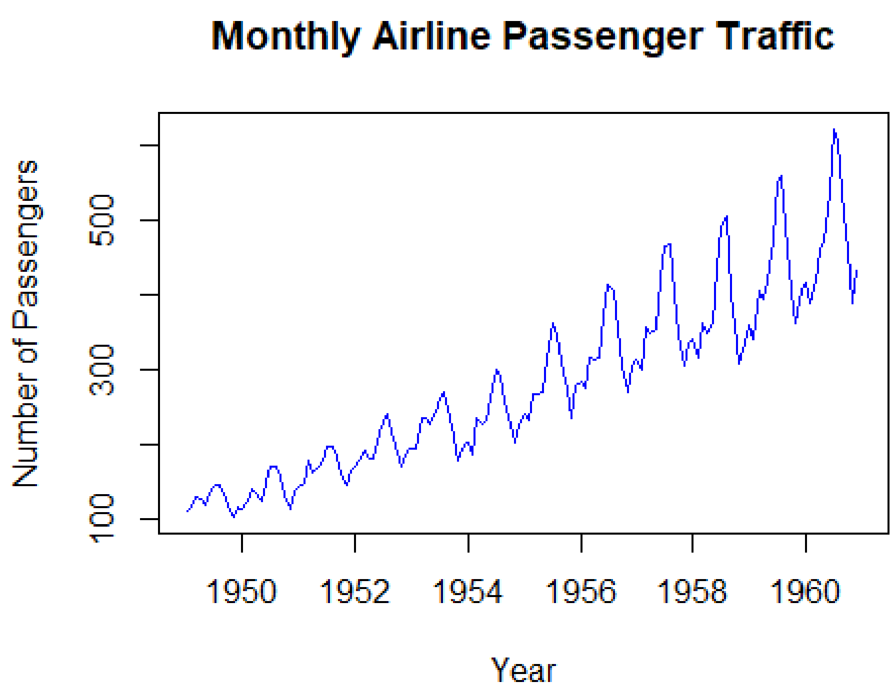
```r
# Summary statistics
summary(data)
```

```
##     Month
##  Length:145
##  Class :character
##  Mode  :character
##
##
##
##
##
International.airline.passengers..monthly.totals.in.thousands..Jan.49...Dec.6
0
##  Min.   :104.0
##  1st Qu.:180.0
##  Median :265.5
##  Mean   :280.3
##  3rd Qu.:360.5
##  Max.   :622.0
##  NA's   :1
```

Step 2: Convert to Time Series Object

```r
# Rename the column for clarity
colnames(data) <- c("Month", "Passengers")
```

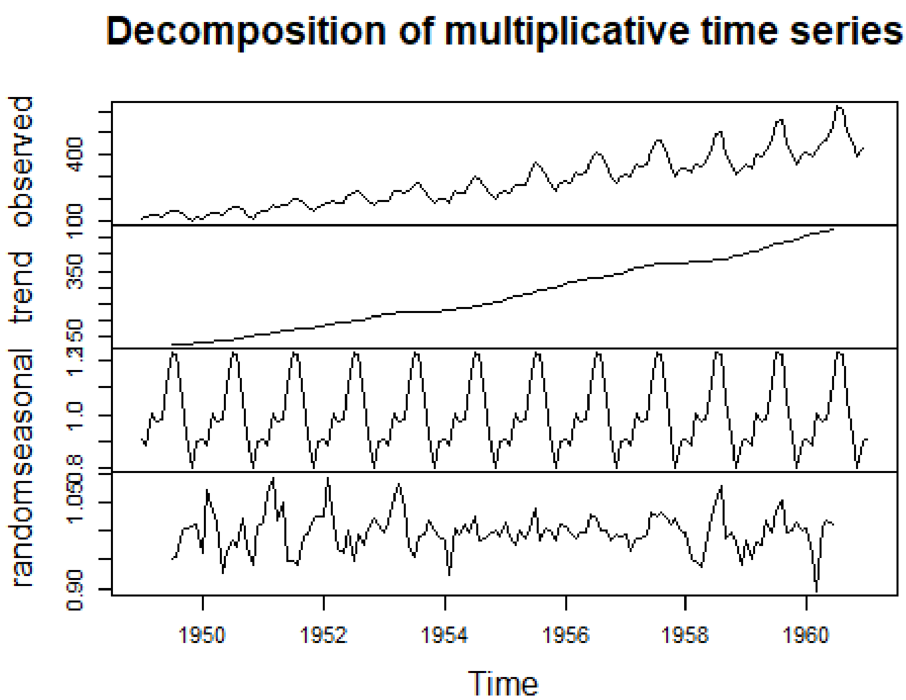
```r
# Convert 'Month' to Date format
data$Month <- as.Date(data$Month, format = "%Y-%m")

# Create the time series object
passenger_ts <- ts(data$Passengers, start = c(1949, 1), frequency = 12)

# Plot the time series
plot(passenger_ts, main = "Monthly Airline Passenger Traffic",
     xlab = "Year", ylab = "Number of Passengers", col = "blue")
```




```r
# Check for missing values
sum(is.na(data$Passengers))

## [1] 1

# Ensure the Passengers column is numeric
data$Passengers <- as.numeric(data$Passengers)

# Verify the structure
str(data)

## 'data.frame':    145 obs. of  2 variables:
##  $ Month     : Date, format: NA NA ...
##  $ Passengers: num  112 118 132 129 121 135 148 148 136 119 ...
```

Step 3: Decompose the Time Series

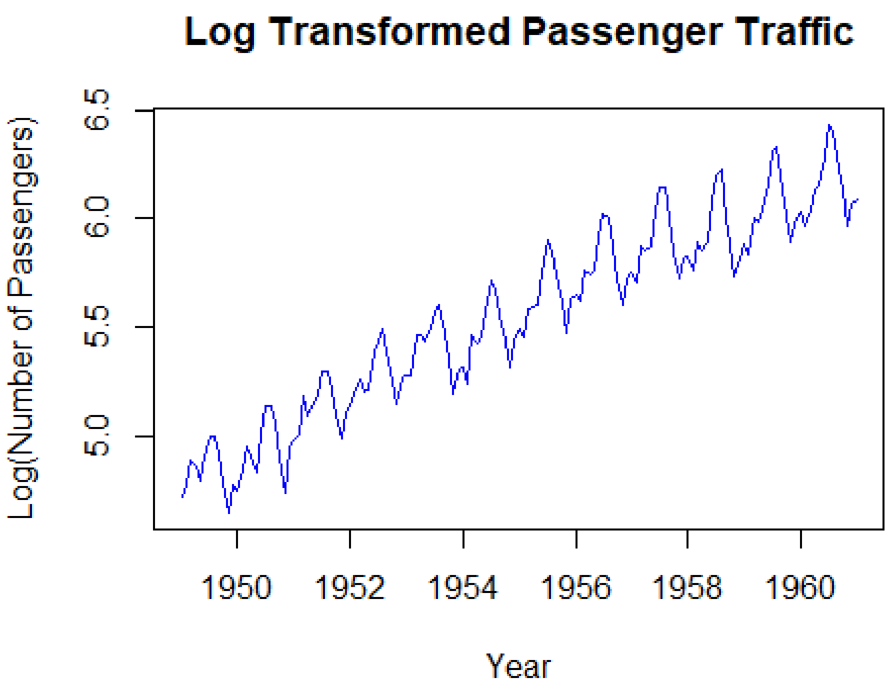
```
# Decompose the time series
decomposed <- decompose(passenger_ts, type = "multiplicative")

# Plot the decomposition
plot(decomposed)
```



Step 4: Check for Stationarity

```
# Check for zeros or NAs in the time series
sum(is.na(passenger_ts))   # Count NA values

## [1] 1

sum(passenger_ts == 0)     # Count zero values

## [1] NA

# Interpolate missing values
library(forecast)

## Warning: package 'forecast' was built under R version 4.4.2

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

passenger_ts <- na.interp(passenger_ts)
```

```r
# Apply log transformation
log_passenger_ts <- log(passenger_ts)

# Verify the log-transformed series
plot(log_passenger_ts, main = "Log Transformed Passenger Traffic",
     xlab = "Year", ylab = "Log(Number of Passengers)", col = "blue")
```




```r
# Perform ADF test
library(tseries)

## Warning: package 'tseries' was built under R version 4.4.2

adf_test <- adf.test(log_passenger_ts, alternative = "stationary")

## Warning in adf.test(log_passenger_ts, alternative = "stationary"): p-value
## smaller than printed p-value

print(adf_test)

##
##  Augmented Dickey-Fuller Test
##
## data:  log_passenger_ts
## Dickey-Fuller = -6.3982, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

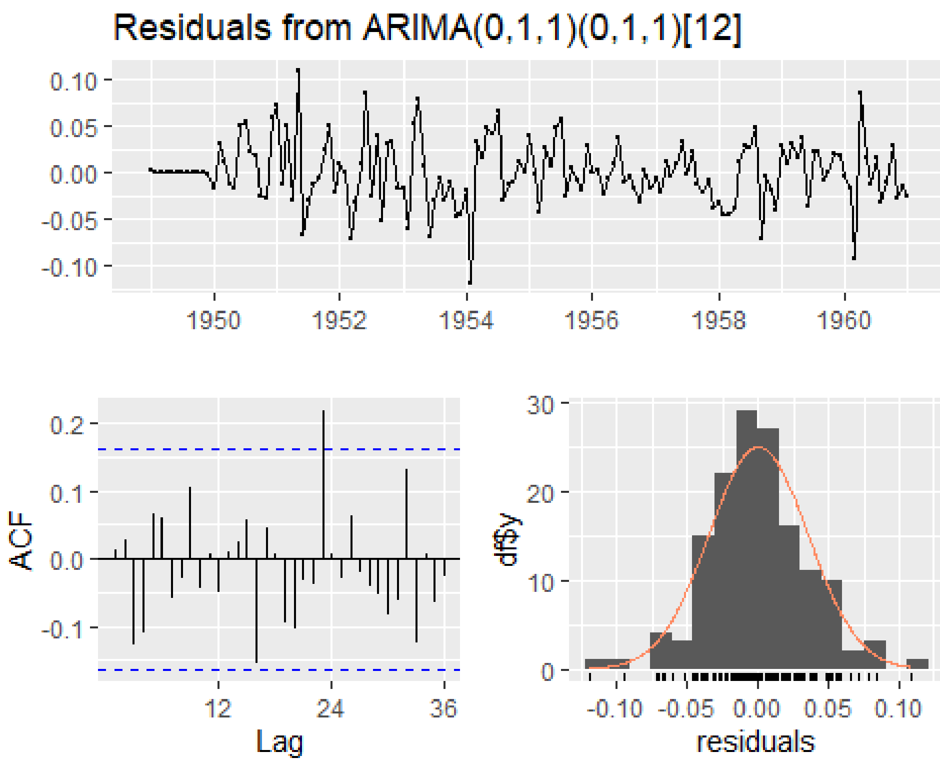
Step 5: Fit a SARIMA Model

```r
# Load the forecast library
library(forecast)

# Fit the SARIMA model
sarima_model <- auto.arima(log_passenger_ts, seasonal = TRUE)

# Print the summary of the fitted model
summary(sarima_model)
```

```
## Series: log_passenger_ts
## ARIMA(0,1,1)(0,1,1)[12]
##
## Coefficients:
##           ma1     sma1
##       -0.3983  -0.5577
## s.e.   0.0895   0.0730
##
## sigma^2 = 0.001366:  log likelihood = 246.84
## AIC=-487.68   AICc=-487.49   BIC=-479.03
##
## Training set error measures:
##                         ME       RMSE        MAE         MPE      MAPE
MASE
## Training set 0.0003871913 0.03499213 0.02626068 0.007912188 0.4749857
0.2179003
##                    ACF1
## Training set 0.01347755
```

Step 6: Check Model Diagnostics

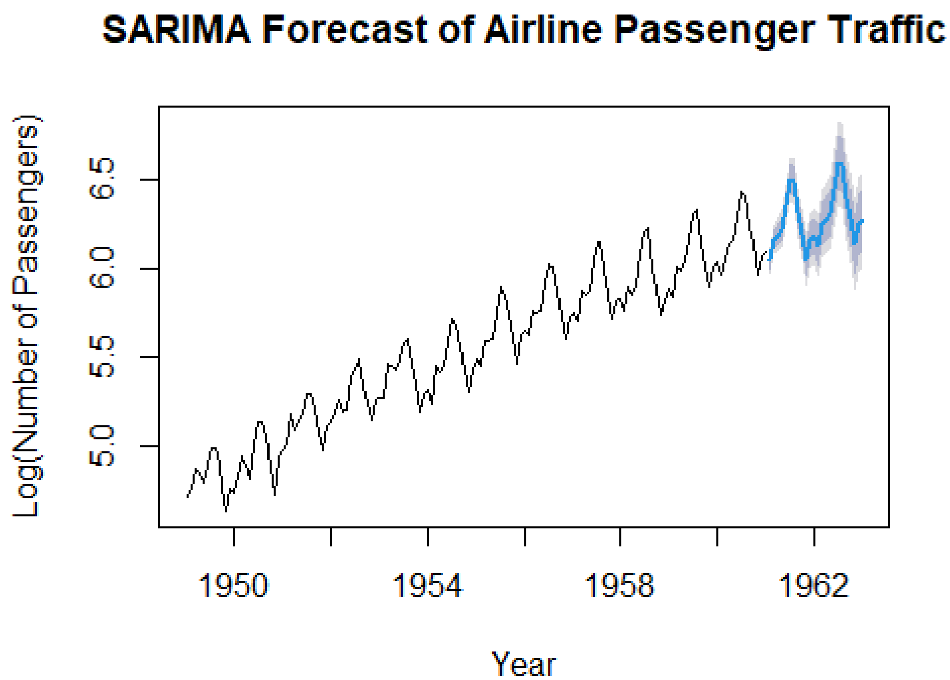
```r
# Plot residual diagnostics
checkresiduals(sarima_model)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(0,1,1)[12]
## Q* = 25.798, df = 22, p-value = 0.2605
##
## Model df: 2.   Total lags used: 24
```

Step 7: Forecast Future Values

```r
# Forecast the next 24 months
forecast_values <- forecast(sarima_model, h = 24)

# Plot the forecast
plot(forecast_values, main = "SARIMA Forecast of Airline Passenger Traffic",
     xlab = "Year", ylab = "Log(Number of Passengers)")
```




```r
# Convert forecast back to the original scale
forecast_values$mean <- exp(forecast_values$mean)

# Print forecasted values
print(forecast_values)

##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Feb 1961       419.1327 5.990827 6.085548 5.965755 6.110620
## Mar 1961       471.6275 6.100917 6.211462 6.071657 6.240721
## Apr 1961       484.7735 6.121496 6.245867 6.088577 6.278787
## May 1961       501.1607 6.148523 6.285331 6.112312 6.321541
## Jun 1961       574.3120 6.279071 6.427275 6.239844 6.466502
## Jul 1961       659.6126 6.412261 6.571045 6.370233 6.613072
## Aug 1961       656.7351 6.402930 6.571632 6.358277 6.616284
## Sep 1961       549.5520 6.220069 6.398138 6.172937 6.445270
## Oct 1961       489.4998 6.099901 6.286867 6.050414 6.336354
## Nov 1961       423.2187 5.950159 6.145619 5.898424 6.197354
## Dec 1961       469.8616 6.050639 6.254238 5.996749 6.308127
## Jan 1962       482.6952 6.073673 6.285098 6.017712 6.341059
## Feb 1962       458.3117 6.010846 6.244253 5.949067 6.306032
## Mar 1962       515.7136 6.121823 6.369280 6.056325 6.434778
## Apr 1962       530.0884 6.142668 6.403420 6.073651 6.472436
## May 1962       548.0075 6.169589 6.442989 6.097224 6.515354
## Jun 1962       627.9967 6.299790 6.585280 6.224226 6.660844
## Jul 1962       721.2709 6.432471 6.729558 6.353837 6.808192
## Aug 1962       718.1244 6.422519 6.730767 6.340930 6.812355
## Sep 1962       600.9222 6.238956 6.557975 6.154517 6.642414
```

```
## Oct 1962       535.2565 6.118027 6.447465 6.030830 6.534662
## Nov 1962       462.7797 5.967482 6.307020 5.877612 6.396890
## Dec 1962       513.7826 6.067127 6.416473 5.974661 6.508939
## Jan 1963       527.8159 6.089305 6.448190 5.994313 6.543181
```